Hardware-Accelerated Networks at Scale in the Cloud

Daniel Firestone

Tech Lead and Manager, Azure Host Networking Group

Microsoft Azure

I’d like to sell you some offloads

Stop me if you’ve heard any of these quotes…

"SR-IOV eliminates virtualization overhead, so I don't need to worry about the 'virtualization tax' of running my workloads in VMs"

"RDMA eliminates the software overhead of TCP, giving me low latency / low jitter / high throughput with 0 CPU"

"PFC, along with my favorite congestion control algorithm, makes my network lossless so I don't have to worry about jitter / restransmits"

"DPDK makes packet processing so cheap, I can do whatever I want on the CPU and ignore the rest of the quotes above"

A nice demo, for example

Microsoft Azure

This talk is NOT just about cool one-box offloads (though they are important, as a starting point)

Instead, a different question: What does it take to make these technologies work at scale in the cloud?

- How do we make these technologies operate at 4-5 9s?
- How do we monitor and detect failures? Automatically recover?
- How do we isolate failures and prevent cascading failures?
- How do we achieve agility and flexibility when we are constantly evolving the network and adding new SDN features?

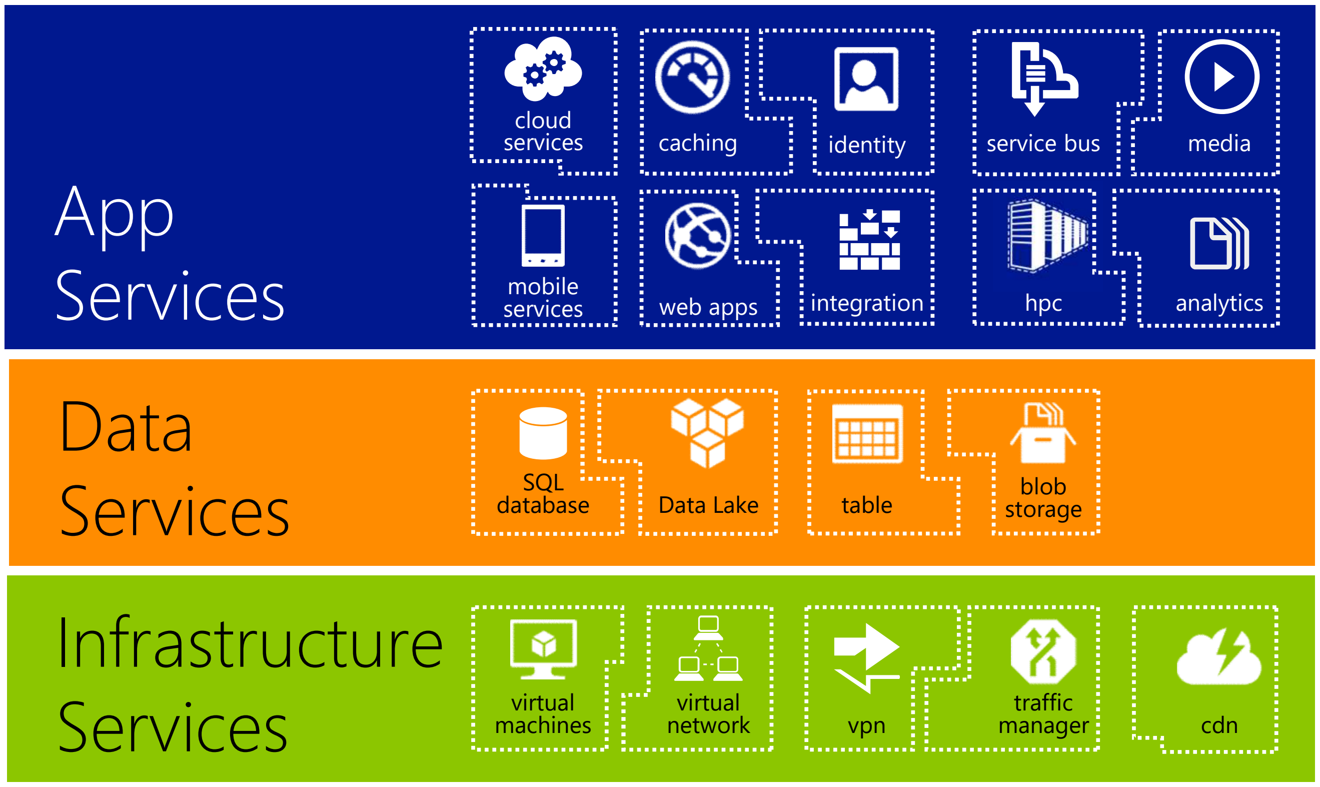
What follows are our some of experiences doing this at scale in Azure… Your mileage may vary

Microsoft Azure

Agenda

- Azure Background
- What does scale mean in cloud?
- Challenges in Virtualization
  - SR-IOV and the pitfalls of “all or nothing” offloads
  - Programmable hardware models
  - Azure SmartNIC – balancing SDN flexibility with hardware performance
  - Virtualizing DPDK
- Challenges in Storage
  - Is DPDK enough?
  - Making RDMA work
  - PFC: Savior of packets or destroyer of networks?
- Closing thoughts

Microsoft Azure



Microsoft Azure

What is Scale?

2013

2014

2015

2016

Coming Soon

Microsoft Azure

	2010	2017
Compute Instances	100K	Millions
Azure Storage	10’s of PB	Exabytes
Datacenter Network	10’s of Tbps	Pbps

Microsoft Azure

Azure Scale & Momentum

>85%
Fortune 500 using Microsoft Cloud

>120,000
New Azure customers a month

>9 MILLION
Azure Active Directory Orgs

>18 BILLION
Azure Active Directory authentications/week

> 3 TRILLION
Azure Event Hubs events/week

>60 TRILLION
Azure storage objects

>900 TRILLION
requests/day

1 out of 3 Azure VMs are Linux VMs

>110 BILLION
Azure DB requests/day

Microsoft Azure

Other ways to think about scale

- You will have device failures, link failures, server failures all the time, multiple at a time
- You will have gray failures as well
  - Lossy links, switches dropping packets greater than x bytes, undetected corruption, etc
- You can (and must) build great automation and highly available designs to detect and repair/remove such failures from the network, and you will still have ones your automation doesn't find
- IaaS customers (e.g. Enterprises) are not tolerant of single VM failures – they expect 4-5 9s of availability
- Must be able to service all parts of the network (including the host) and still achieve this availability
- Performance, availability, serviceability downtimes, are all measured by P99/P99.9, not P50

Microsoft Azure

Virtualization

IaaS is at the core of Cloud…

Microsoft Azure

Why do we need Virtual Switch SDN Policy?

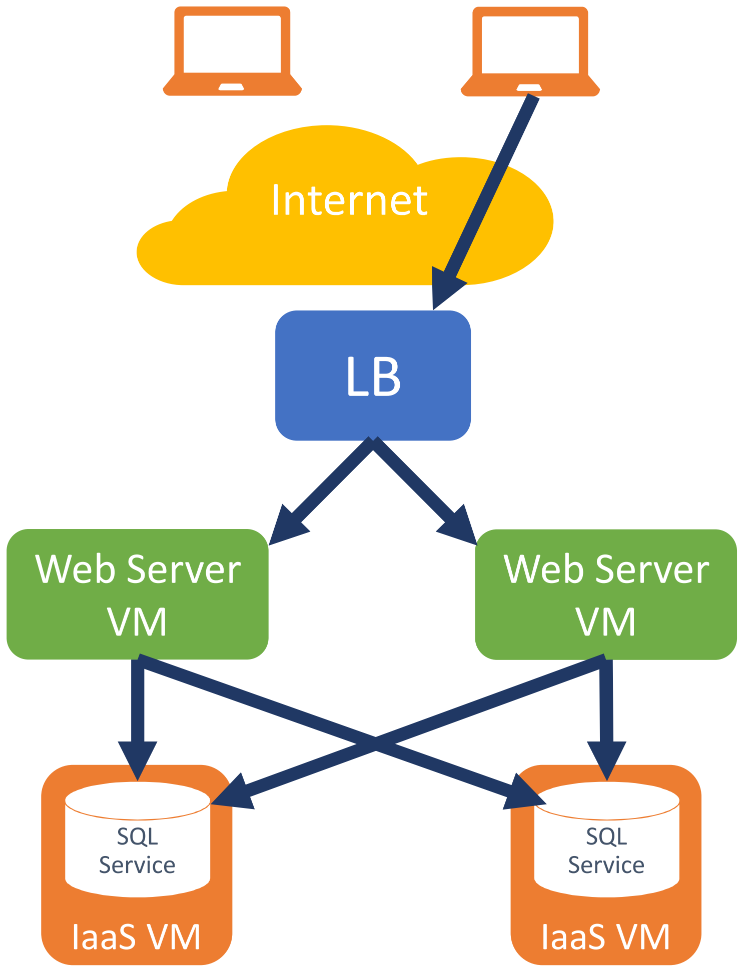
Policy application at the host is more scalable!

Microsoft Azure

Example #1: LB (From Ananta, SIGCOMM '13)

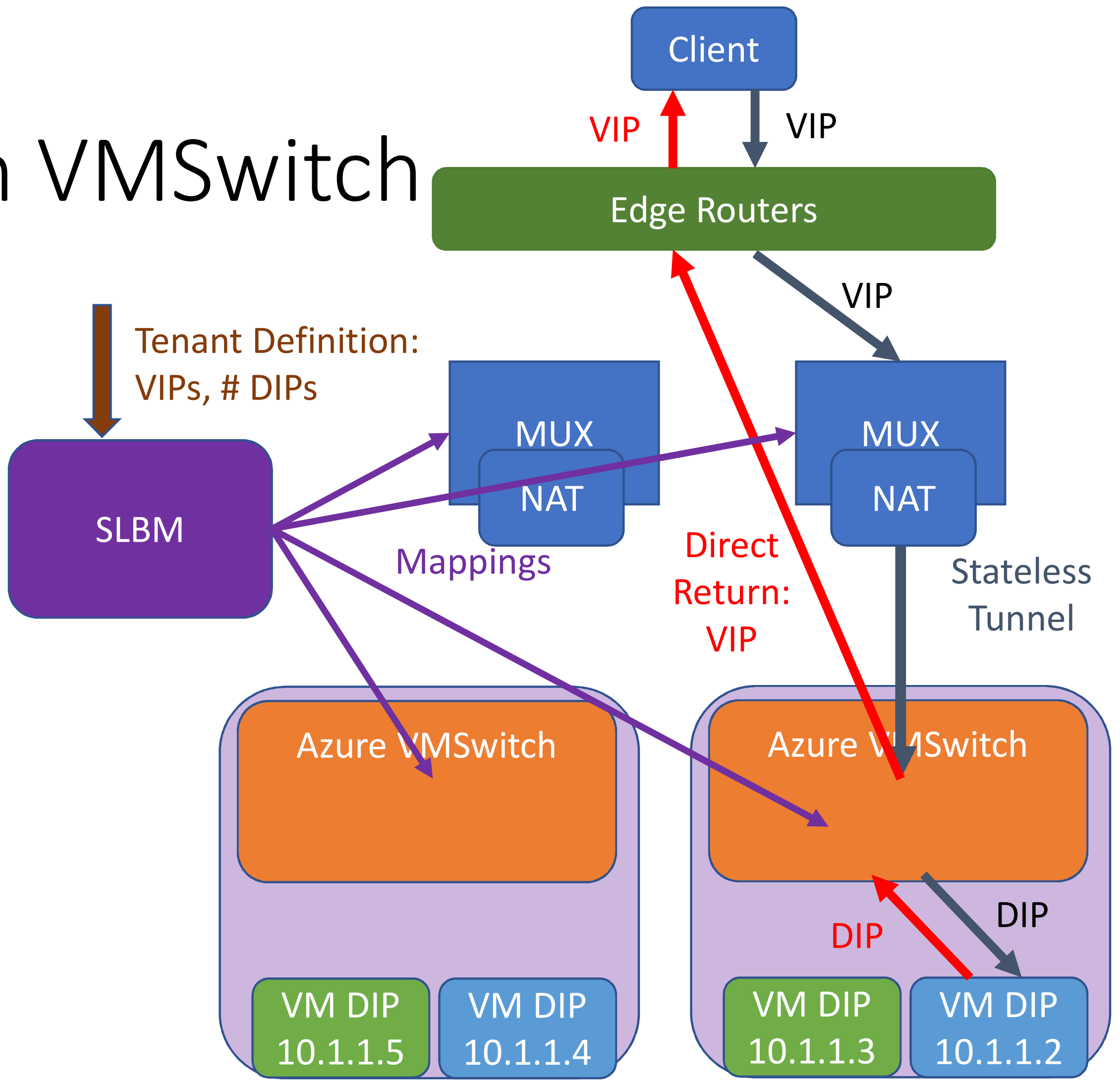
- All infrastructure runs behind an LB to enable high availability and application scale
- How do we make application load balancing scale to the cloud?
- Challenges:
    - How do you load balance the load balancers?
    - Hardware LBs are expensive, and cannot support the rapid creation/deletion of LB endpoints required in the cloud
    - Support 10s of Gbps per cluster
    - Need a simple provisioning model

Microsoft Azure



"SDN" Approach: Software LB with NAT in VMSwitch

- Goal of an LB: Map a Virtual IP (VIP) to a Dynamic IP (DIP) set of a cloud service
- Two steps: Load Balance (select a DIP) and NAT (translate VIP->DIP and ports)
- Pushing the NAT to the vswitch makes the MUXes stateless (ECMP) and enables direct return
- Single controller abstracts out LB/vswitch interactions



Microsoft Azure

Example #2: Vnet

- Ideas from VL2 (SIGCOMM '09)
- Goal is to map Customer Addresses (e.g. BYO IP space) to Provider Addresses (real 10/8 addresses on the physical network)
- This requires a translation of *every* packet on the network – no hardware device on our network is scalable enough to handle this load along with all of the relevant policy
- Enables companies to create their own virtual network in the cloud, defining their own topologies, security groups, middleboxes and more

Microsoft Azure

Flow tables: The right abstraction for the host

VSwitch exposes a typed Match-Action-Table API to the controller

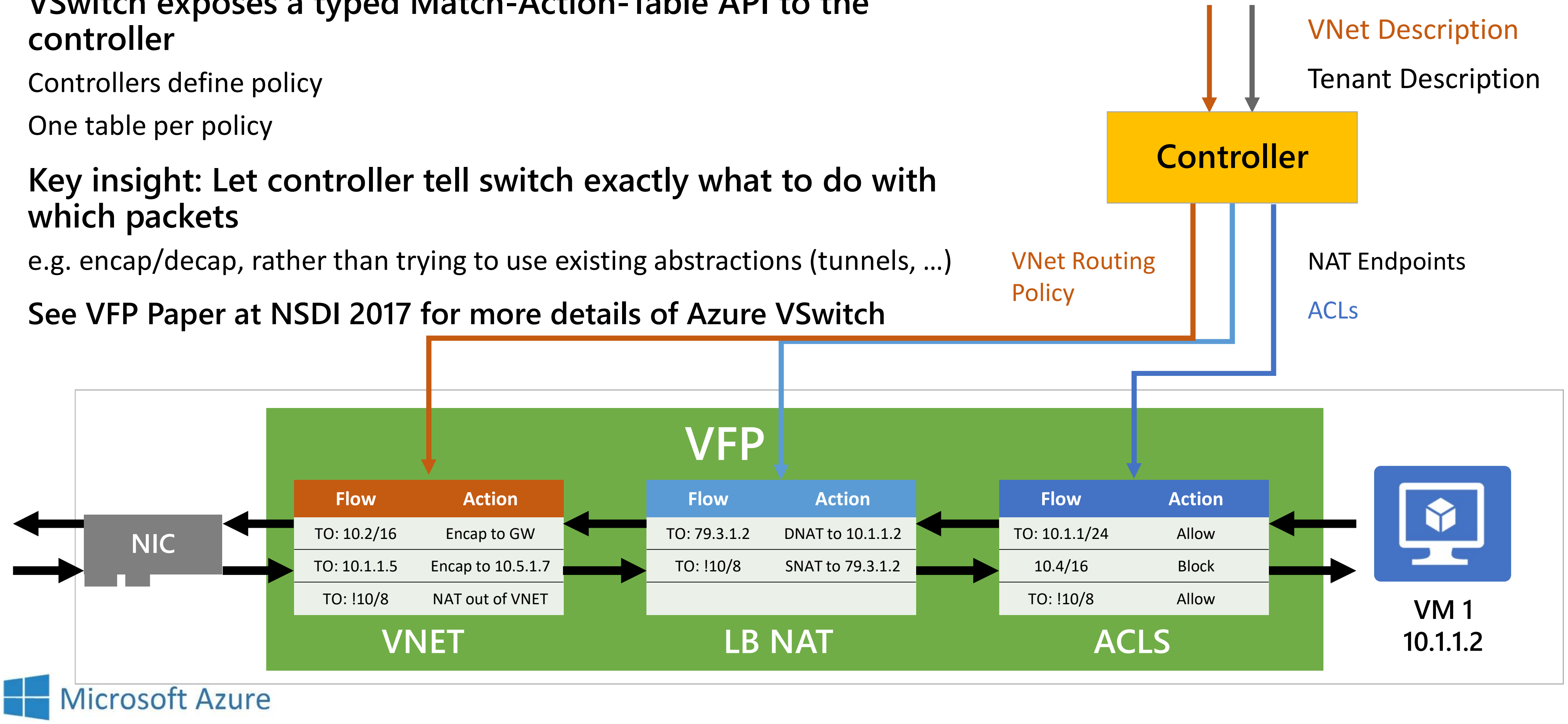
Controllers define policy

One table per policy

Key insight: Let controller tell switch exactly what to do with which packets

e.g. encap/decap, rather than trying to use existing abstractions (tunnels, …)

See VFP Paper at NSDI 2017 for more details of Azure VSwitch



VNET

Flow	Action
TO: 10.2/16	Encap to GW
TO: 10.1.1.5	Encap to 10.5.1.7
TO: !10/8	NAT out of VNET

LB NAT

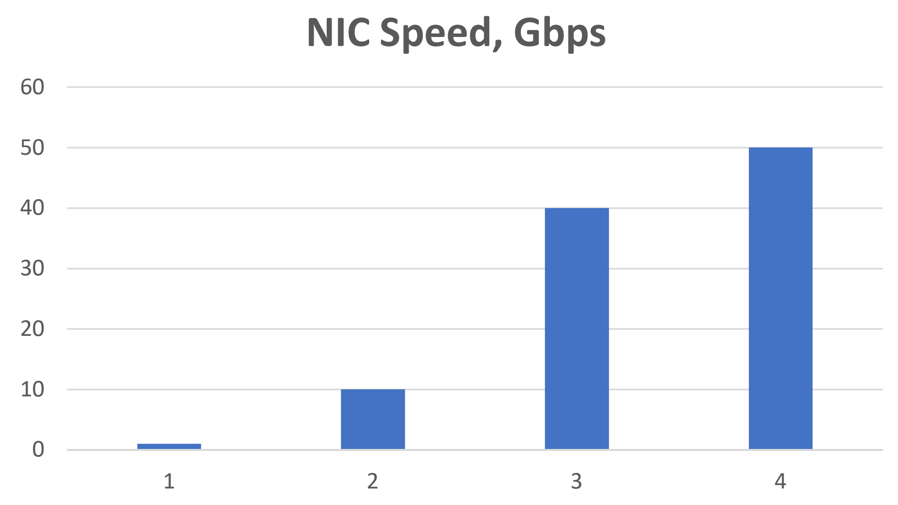
Flow	Action
TO: 79.3.1.2	DNAT to 10.1.1.2
TO: !10/8	SNAT to 79.3.1.2

ACLS

Flow	Action
TO: 10.1.1/24	Allow
10.4/16	Block
TO: !10/8	Allow

Scaling Up SDN: NIC Speeds in Azure

- 2009: 1Gbps
- 2012: 10Gbps
- 2015: 40Gbps
- 2017: 50Gbps
- Soon: 100Gbps?



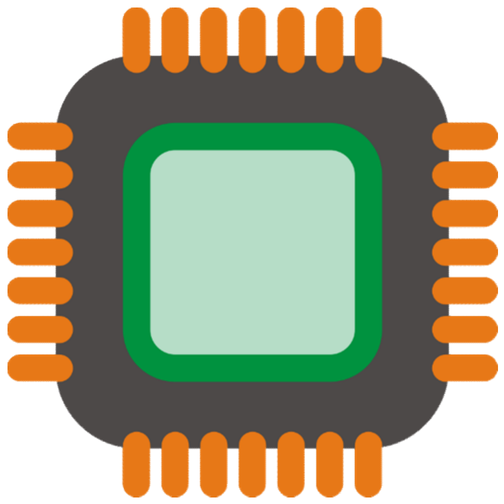
We got a 50x improvement in network throughput, but not a 50x improvement in CPU power!

Microsoft Azure

Host SDN worked well at 1GbE, ok at 10GbE… what about 40GbE+?

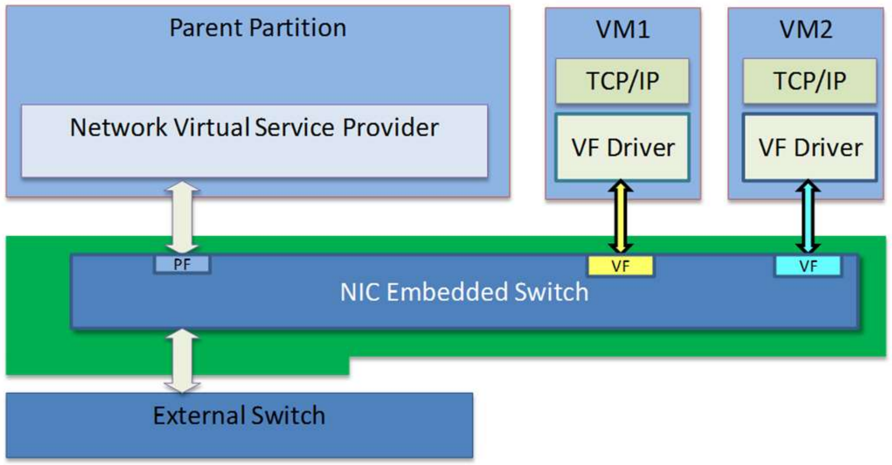
Traditional Approach to Scale: ASICs

- We've worked with network ASIC vendors over the years to accelerate many functions, including:
  - TCP offloads: Segmentation, checksum, …
  - Steering: VMQ, RSS, …
  - Encapsulation: NVGRE, VXLAN, …
  - Direct NIC Access: DPDK, PacketDirect, …
  - RDMA
- Is this a long term solution?



Microsoft Azure

Example ASIC Solution: Single Root IO Virtualization (SR-IOV) gives native performance for virtualized workloads



But where is the SDN Policy?

Microsoft Azure

Hardware or Bust

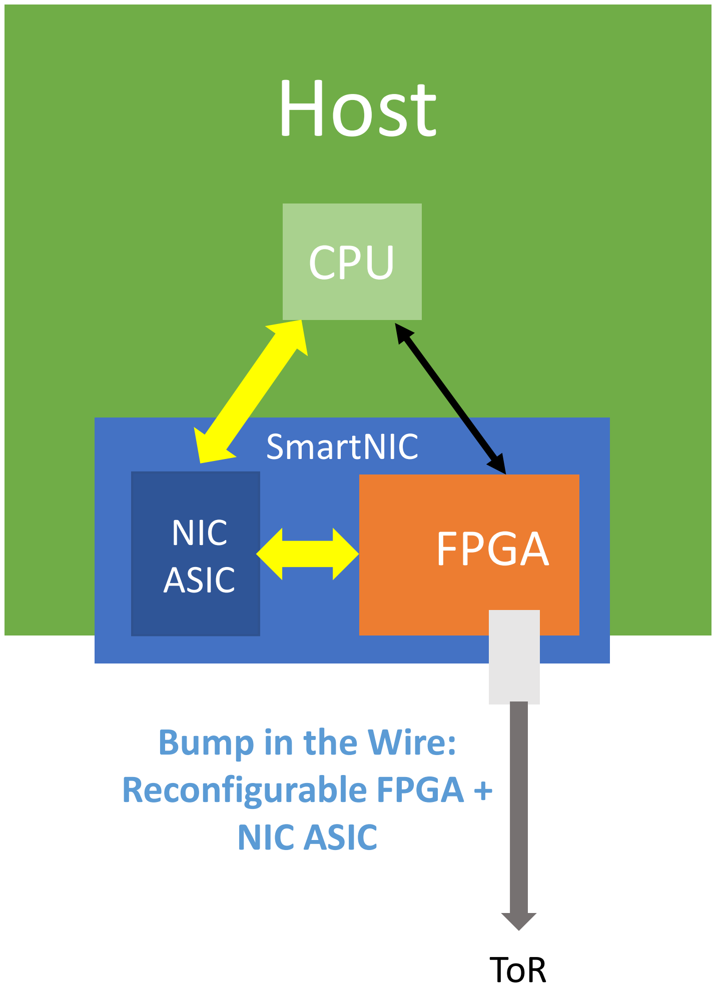
- SR-IOV is a classic example of an “all or nothing” offload – its latency, jitter, CPU, performance benefits come from skipping the host entirely
- If even one widely-used action isn’t supported in hardware, have to fall back to software path and most of the benefit is lost even if hardware can do 99% of the work
- Other examples: RDMA, DPDK, … a common pattern
- This means we need to consider carefully how we will add new functionality to our hardware as needed over time

How do we get the performance of hardware with programmability of software?

Microsoft Azure

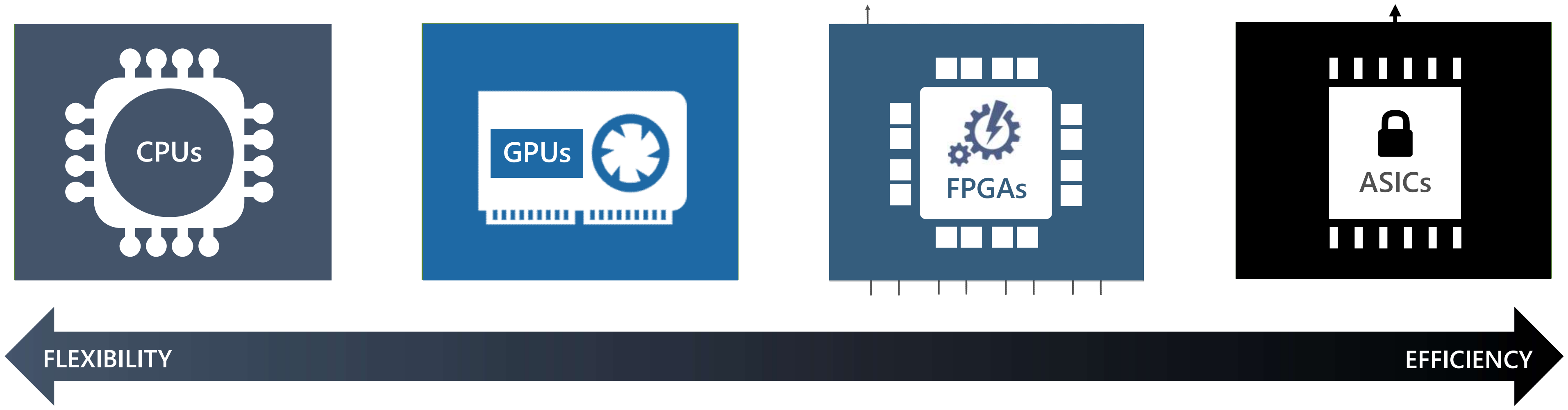
Our Solution:
Azure SmartNIC (FPGA)

- HW is needed for scale, perf, and COGS at 40G+
- 12-18 month ASIC cycle + time to roll new HW is too slow
- To compete and react to new needs, we need agility – SDN
- Programmed using Generic Flow Tables
  - Language for programming SDN to hardware
  - Uses connections and structured actions as primitives



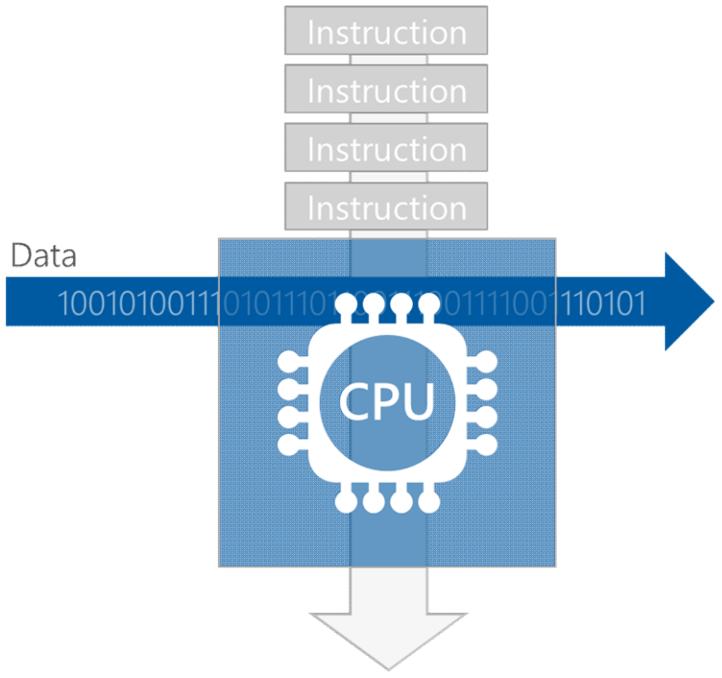
Microsoft Azure

Silicon alternatives



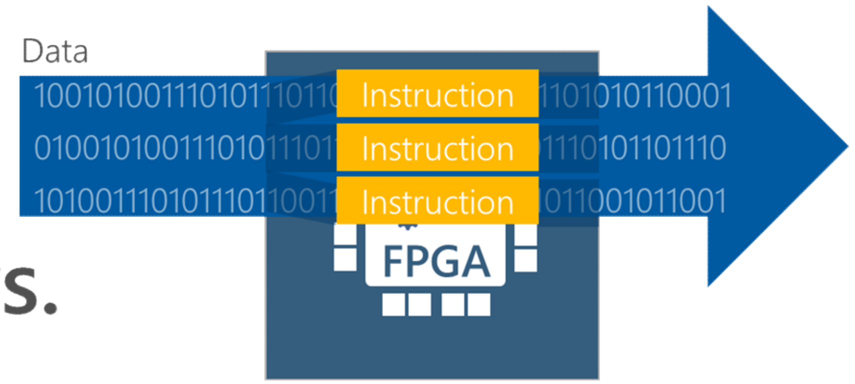
Microsoft Azure

CPU vs. FPGA



CPU: temporal compute

vs.

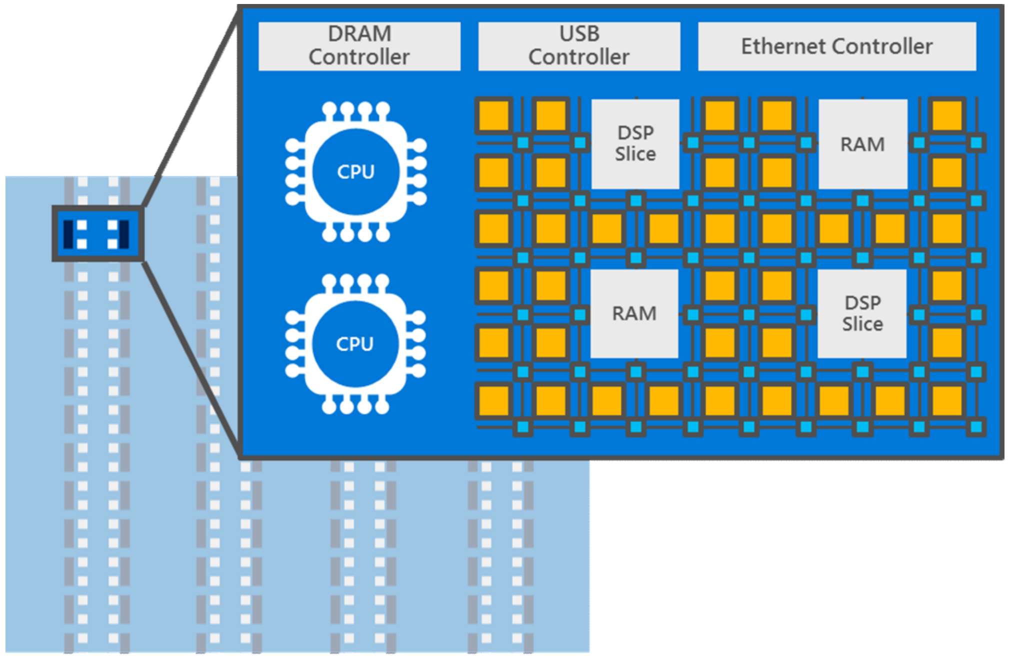


FPGA: spatial compute

Microsoft Azure

What is an FPGA, Really?

- Field Programmable Gate Array
- Chip has large quantities of programmable gates – highly parallel
- Program specialized circuits that communicate directly
- Two kinds of parallelism:
  - Thread-level parallelism (stamp out multiple pipelines)
  - Pipeline parallelism (create one long pipeline storing many packets at different stages)
- FPGA chips are now large SoCs (can run a control plane)



Microsoft Azure

Programmable Hardware Alternatives

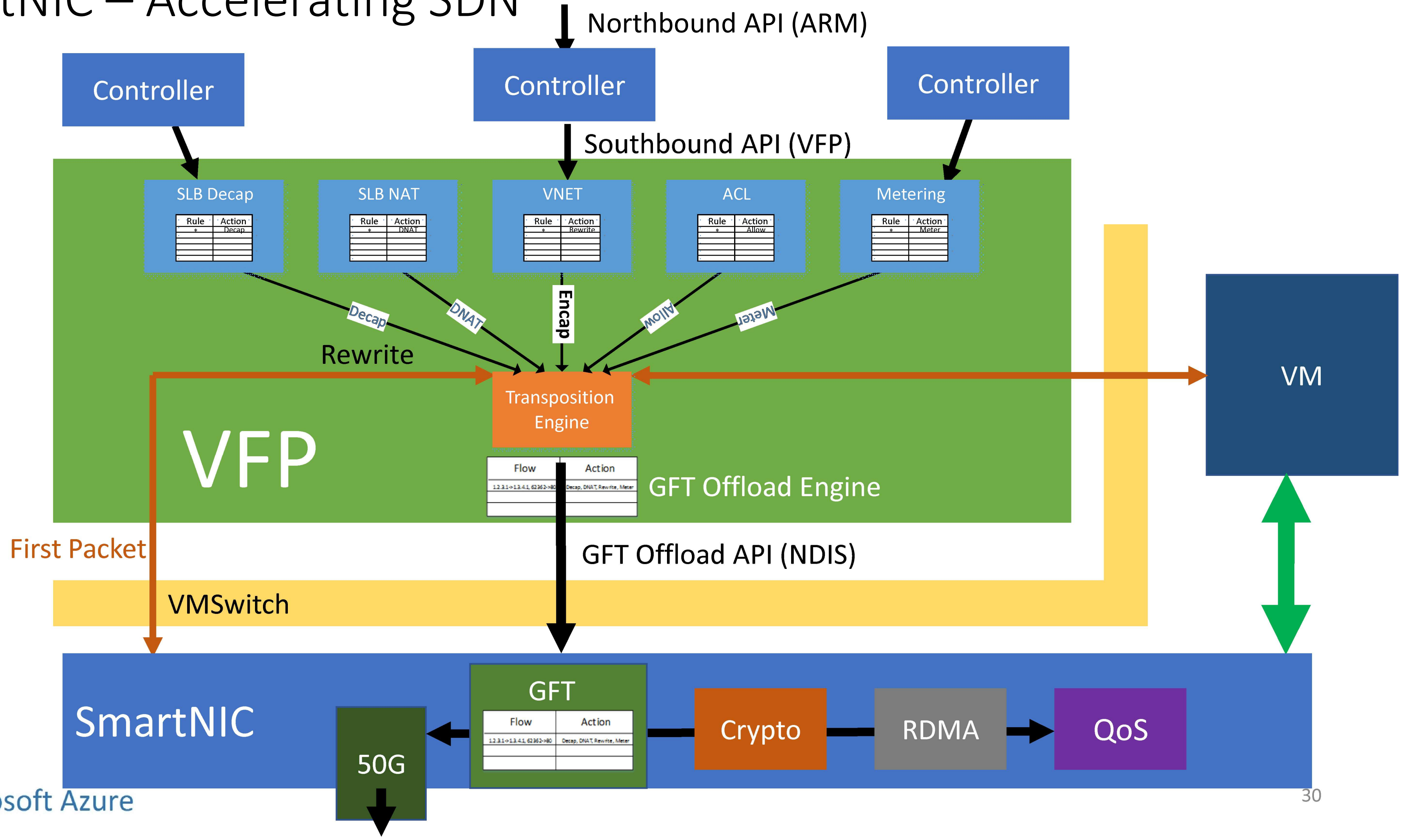
- FPGA Pros
  - Great performance with great programmability
  - With pipeline parallelism, can process line rate at 100Gbps+ with small packets in a reconfigurable pipeline
- FPGA Cons
  - Need to understand hardware arch and digital logic design – code in Verilog
  - Cost of logic per unit performance higher than ASIC (though lower than GP cores)
- Core-based NICs
  - Relatively easy to program in C, but performance is limited and parallelism is very course – difficult to scale up to 100Gbps+ without compromises
- ASIC-based NICs
  - Can implement efficient MATs, but with limited flexibility
  - P4 and P4-based NICs not expressive enough to create efficient general purpose pipelines
- Note: while packet-processing logic is cheaper on ASICs, on-chip memory / IO / DRAM / storage / SoC cores are not – total system cost is not necessarily that different

Azure Accelerated Networking: Fastest Cloud Network!

- Highest bandwidth VMs of any cloud
  - Standard compute (D series) VMs get 25Gbps
  - Big compute (M series) gets 32Gbps
  - Standard Linux VM with CUBIC gets 30+Gbps on a single connection
- Consistent low latency network performance
  - Provides SR-IOV to the VM
  - Up to 10x latency improvement – sub 25us within VM Scale Sets
  - Increased packets per second (PPS)
  - Reduced jitter means more consistency in workloads
- Enables workloads requiring native performance to run in cloud VMs
  - >2x improvement for many DB and OLTP applications

Microsoft Azure

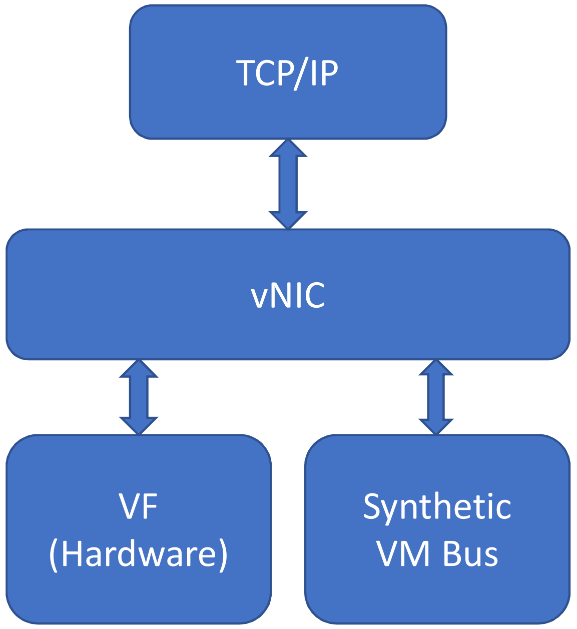
SmartNIC – Accelerating SDN



Microsoft Azure

Serviceability is Key

- All parts of this system can be updated, any of which require us to take out the hardware path
    - FPGA image, driver, GFT layer, Vswitch/VFP, NIC PF driver
- IaaS requires high uptime and low disruption – can't take away the NIC device from under the app, and can't reboot the VM / app
- Instead, we keep the synthetic vNIC and support transparent failover between the vNIC and VF

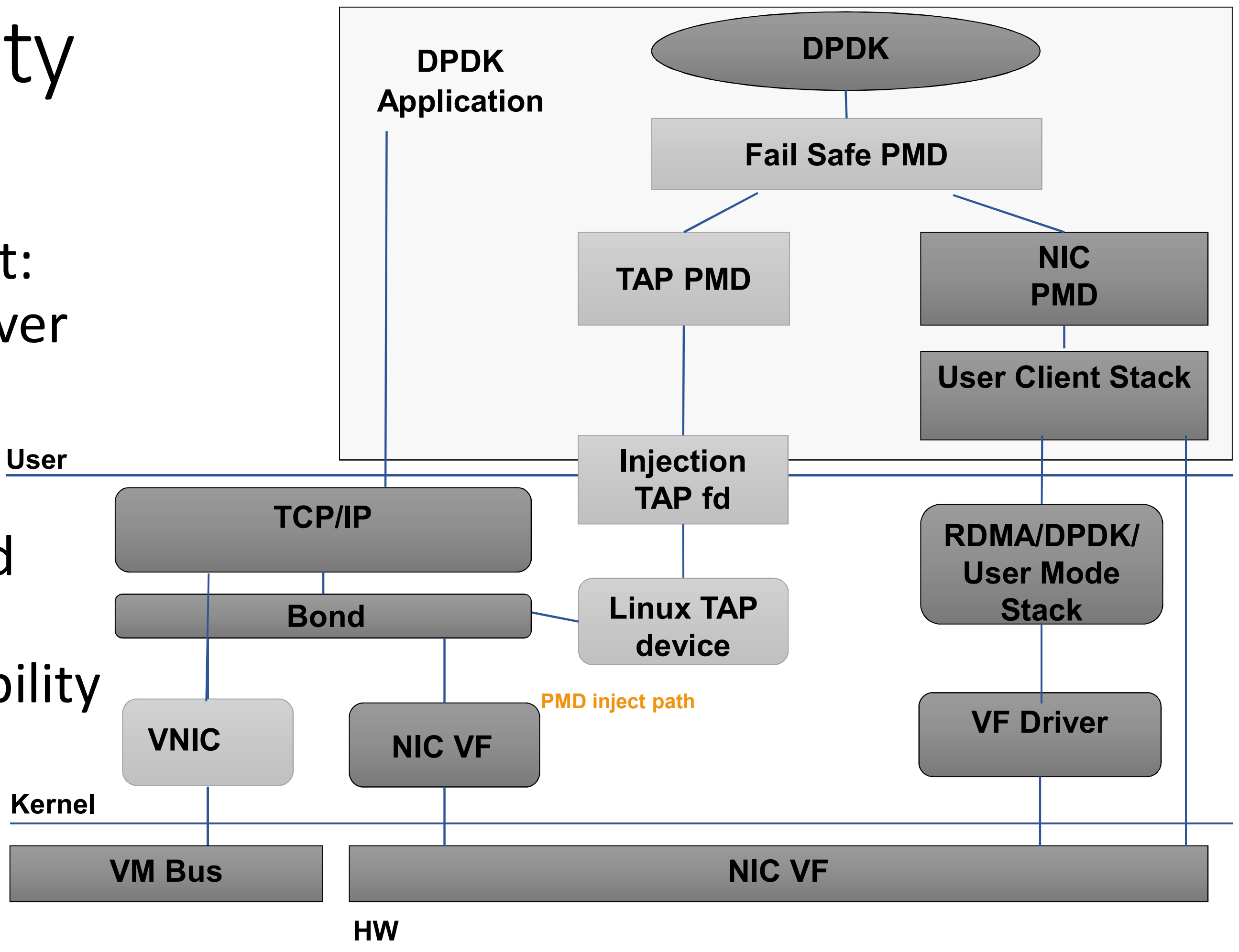


Lesson: A huge amount of the effort to deploy SR-IOV was in making all parts of this path rebootlessly serviceable without impact

Microsoft Azure

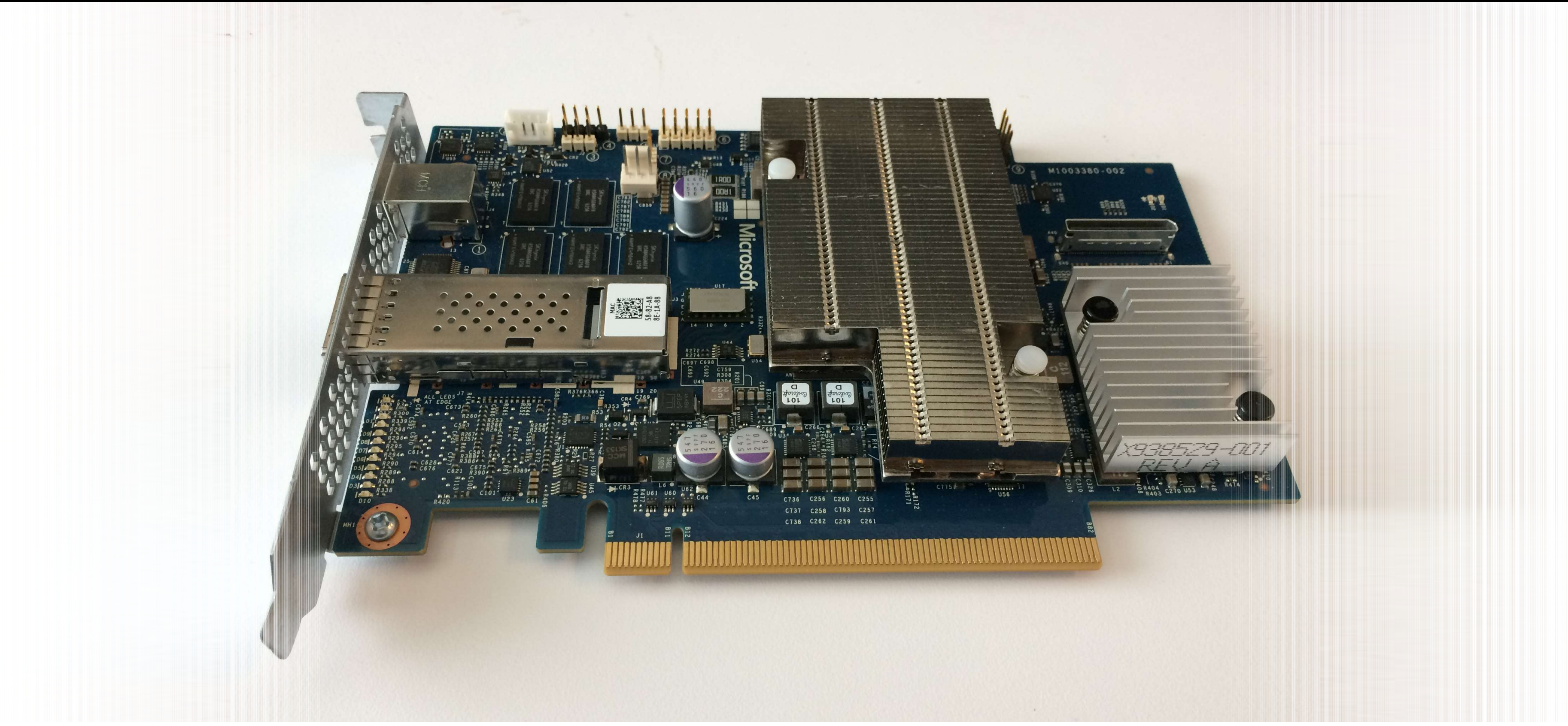
DPDK Serviceability

- Extend the bonding concept: Create a software DPDK driver to failover from hardware
- DPDK app is not interrupted (except for performance) during failover for serviceability



Microsoft Azure

New **50GbE** SmartNIC for Project Olympus
(Announced at OCP 2017)



Storage

Block storage, blob storage, DBs (SQL and NoSQL), Big Data, HDFS/Spark, …

Microsoft Azure

What are our goals?

- Low latency with high IOPS
  - Customers primarily care about jitter and P99.99 latency, not P50 latency
  - Mission critical workloads will pay a lot for small improvements in tail
- Low CPU overhead – Cores == Revenue in the cloud
- High throughput elephant flows
- Efficient small message passing
- Highly tolerant of arbitrary failures at any point in the network
- Operationalization: Great monitoring, visibility, diagnosability, manageability, and “self-healing”

Microsoft Azure

What are not our goals?

- Designing for high utilization of the network
- This is a terrible idea
- The network costs a small fraction of the servers
    - Consider the silicon cost of a single 3.2Tbps switch chip vs a rack of CPUs, DRAM, and Flash
    - It's relatively cheap to provision multiple times as much capacity as average workloads require
- Highly utilized networks are very brittle
    - Small numbers of coordinated failures can be devastating
    - Average utilization >30% means you have spikes/congestion 100% of the time

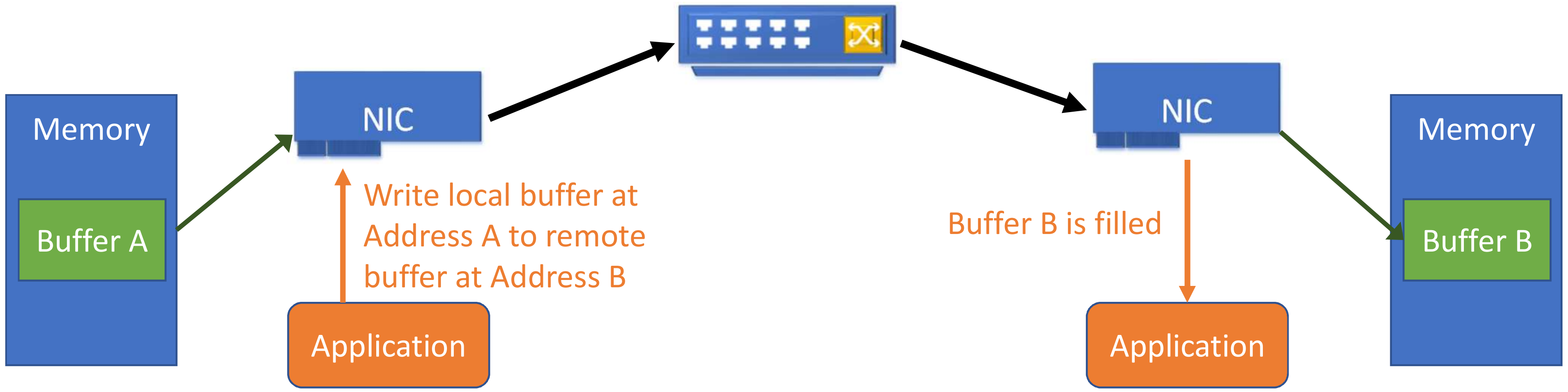
High utilization kills P99.99 IO completion in real networks and doesn't save much cost

Throwing cores at the problem

Why not use DPDK to reduce the cost of the transport layer?

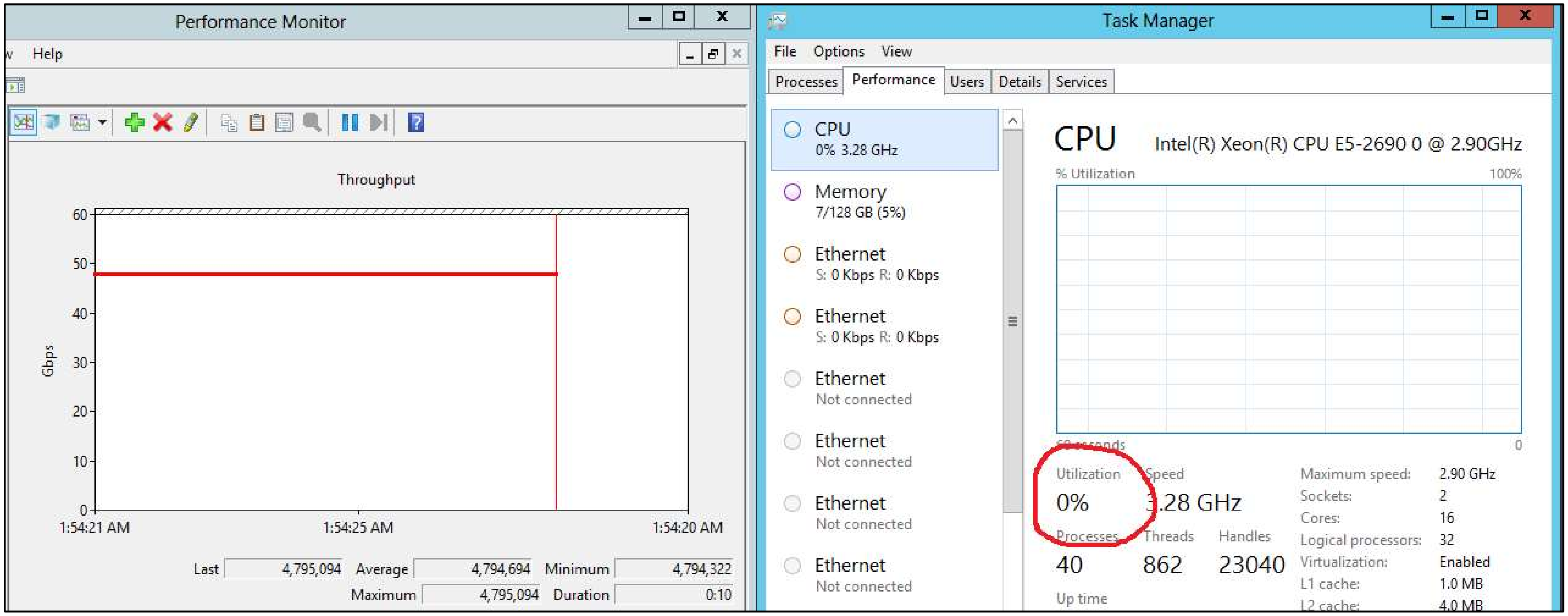
1. The network isn't that expensive, but cores are – standard 2-socket blades' VM density is limited by cores (can always add DRAM/flash)
2. Scales poorly for elephant flows (hard to split a stateful flow across cores)
3. Even poll-mode is subject to the whims of the scheduler and cache – latency and jitter can be high, true core isolation is hard to achieve
4. DPDK is a one time benefit (exposed wildly inefficient data processing that happened over time in Linux/Windows kernels to enable extensibility). But NIC throughput is still growing faster than CPUs are adding cores or than cores are getting faster – the trendline is not good over time (100G? 200G? 400G?)

RDMA – High Performance Transport



- Remote DMA primitives (e.g. Read address, Write address) implemented on-NIC
  - Zero Copy (NIC handles all transfers via DMA)
  - **<u>Zero CPU Utilization at 40Gbps</u>** (NIC handles all packetization)
  - <2µs E2E latency
- RoCEv2 enables Infiniband RDMA transport over IP/Ethernet network (all L3)
- Enabled at 40GbE for Azure Storage, achieving large COGS savings by eliminating many CPUs in the rack

Yes, it's real…
40Gbps of I/O with 0% CPU



Last 4,795,094 Average 4,794,694 Minimum 4,794,322
Maximum 4,795,094 Duration 0:10

CPU 0% 3.28 GHz
Memory 7/128 GB (5%)
Ethernet S: 0 Kbps R: 0 Kbps
Ethernet S: 0 Kbps R: 0 Kbps
Ethernet Not connected
Ethernet Not connected
Ethernet Not connected
Ethernet Not connected

CPU Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz

Utilization	Speed		Maximum speed:	2.90 GHz
0%	3.28 GHz		Sockets:	2
			Cores:	16
Processes	Threads	Handles	Logical processors:	32
40	862	23040	Virtualization:	Enabled
Up time			L1 cache:	1.0 MB
			L2 cache:	4.0 MB

Microsoft Azure

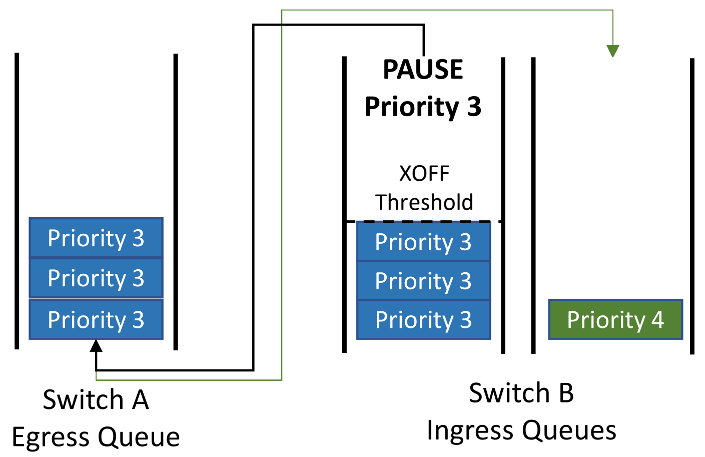
The core technology behind RDMA is great.

What are the pitfalls?

Microsoft Azure

Priority Flow Control (PFC)

- Commonly deployed with RDMA
- A form of hop-by-hop congestion control
- Instead of pausing an entire port, pause a given priority (e.g. Storage) to avoid collateral damage
- Creates a “lossless” network (assuming well behaved network elements…)



Microsoft Azure

PFC: Do we need it?

- PFC has been used as a crutch by NICs that can't handle retransmit flows efficiently. This is bad.
  - See today's RDMA/PFC paper for great ideas on improving this
- But PFC is also a great way to lower jitter in your network if you have a dedicated class of service for critical traffic like storage
  - In conjunction with reasonable congestion control (e.g. DCQCN)
- Thought experiment – if you have identified traffic as a critical storage I/O, and are optimizing for storage I/O completion time, what are the possible benefits to dropping/cutting the packet vs buffering it?
  - You will end up retransmitting the same packet, likely over the same path, and your I/O will now complete later
  - Side note: good luck setting small retransmission timeouts in a dynamic DC network

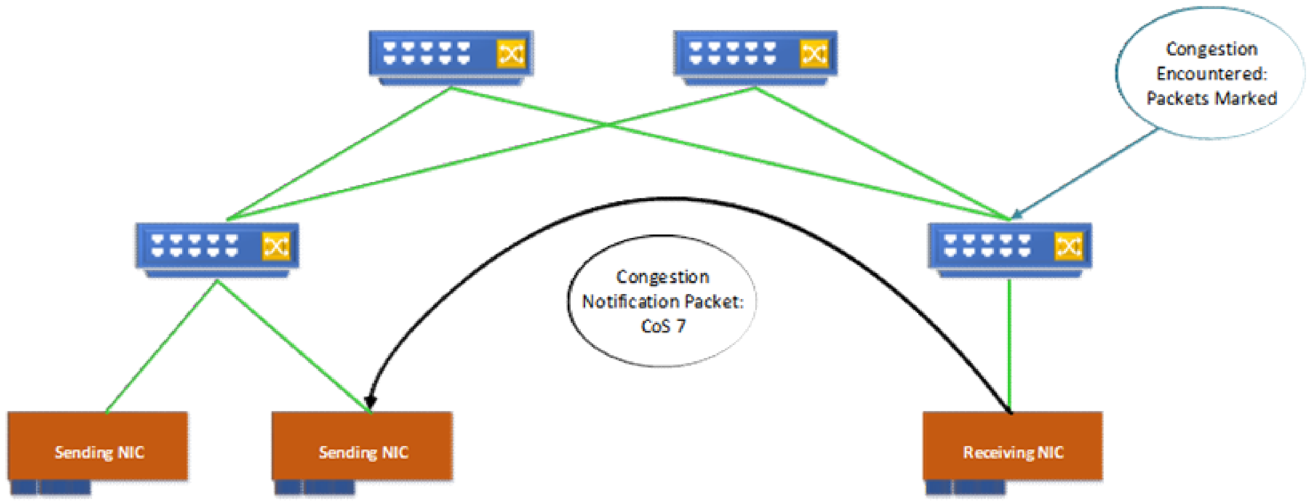
PFC is exactly the technology we want to optimize storage I/O jitter, if we can make it work well…

Microsoft Azure

Making PFC Safe

- PFC is subject to various bad cascading collapses – pause trees, bad switches, bad NICs, bad servers (e.g. asserting constant pause, or draining too slowly)
- In a *well behaved* network, congestion control should keep pause behavior reasonable (e.g. PFC is the ambulance, until CC kicks in)
- Key idea: detect non-well behaved elements in the network and stop enforcing PFC for them
- Isolate failures by disabling PFC for bad peers – each switch and NIC must protect itself from each of its peers (detect stuck queues, consistent near-zero drain)
- Switch to lossy mode for these elements until the problem is solved (hosts need to redial bad connections until they find a working path)

DCQCN: E2E Congestion Control for RRoCE on-NIC (based on Microsoft DCTCP)


Congestion Encountered: Packets Marked
Congestion Notification Packet: CoS 7
Sending NIC
Sending NIC
Receiving NIC

- SIGCOMM 2015
- Explicit Congestion Notification can mitigate unfairness and improve link utilization – standardized in RoCEv2
- Merge DCTCP's quantized congestion estimator with a QCN rate limiter

RDMA has positive feedback loops

- Common functions are typically implemented all in hardware
  - Note: if your RDMA provider admits to implementing their datapath on embedded CPU cores rather than in hardware, this is not usually a good sign
- Unfortunately, functions for handling misbehaving peers are often implemented in firmware
- Firmware flows often have to slow down hardware paths to get things back to a good state
- As a result, bad peers can cause more bad peers, with cascading results
- Your system must be designed to detect bad peers and stop communicating RDMA with them – if redialing connections (new path, new state) doesn't work, must fail over to TCP or fail over to another node until the problem goes away

Microsoft Azure

Must Haves for your Hardware Transport: Service Fundamentals I

- Monitoring – how do we detect RDMA misbehaving
    - Pingmesh (SIGCOMM 2015) can provide good background monitoring for RDMA
    - Scalable metric logging platform for NIC counters, RDMA app, switch counters (PFC)
    - Alerting on a number of thresholds for bad RDMA behavior
- Diagnosability – how do we understand RDMA problems
    - NIC driver + firmware full state dump – triggered by driver or by app automatically in error flows
- Manageability – how do we manage and deploy RDMA changes
    - Dynamic config profiles for NIC driver/firmware, RDMA stack
    - Rebootless driver/firmware update support
    - Config monitoring and update for switches

Must Haves for your Hardware Transport: Service Fundamentals II

- Resiliency – how do we autorecover from failures and fall back to TCP
  - TCP and RDMA are probably not as isolated in your NIC as you'd like them to be (and you won't understand how until you dive deep into your NIC design)
  - Support full stack restart – reset the NIC and bring up from scratch – should guarantee that TCP failover works correctly
  - Note on recovery paths: Never build specialized recovery paths! They never get properly scale tested. Instead, always reset into the normal bringup path
  - Automate resiliency flows on defined triggers in NIC driver/firmware and app
- Automate detection of bad elements and RMA/removal of them

We saw and learned from many failures over time - the vast majority of the work to enable RDMA stably in Azure was on these fundamentals

Microsoft Azure

This is Incredibly Difficult to Properly Operationalize at Scale

But the benefits are enormous…

- COGS
- Performance
- Consistency

No free lunch: It takes a large, dedicated team focused on building good service fundamentals at every layer of the stack

Microsoft Azure

Some Final Thoughts

- Even if you don’t build everything to start, design with service fundamentals in mind from day 1
  - How would I monitor this system, and diagnose issues? How would I service it? How can it detect its own health and recover from failures?
- Datacenters are really cool for research! But lots of knowledge comes from operating a service at scale
  - Consider chatting with or collaborating with your friendly local cloud people – we’re happy to give free advice! Make sure you’re optimizing for the right problems
- Programmable hardware for SDN is an incredible new area for exploration – but you can’t get programmable hardware without hardware!
  - Abstracting away the hardware designs takes away most of what hardware is good at
  - If you want to do programmable hardware research, go learn Verilog and some CompArch – it’s not that hard, and will enable you to solve real problems!
  - Even if you don’t end up a HW dev, this will teach you the real constraints of the system

Microsoft Azure

Microsoft

Want to come build the next generation of scalable Host SDN? We’re hiring!

fstone@microsoft.com

Microsoft Azure

Questions?

Microsoft Azure